Network Working Group E. Rescorla
Internet-Draft RTFM, Inc.
Intended status: Standards Track J. Uberti
Expires: April 13, 2013 Google
E. Ivov
Jitsi
October 10, 2012

Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol

draft-rescorla-mmusic-ice-trickle-00

Abstract

This document describes an extension to the Interactive Connectivity Establishment (ICE) protocol that allows ICE agents to send and receive candidates incrementally rather than exchanging complete lists. With such incremental provisioning, ICE agents can begin connectivity checks while they are still gathering candidates and considerably shorten the time necessary for ICE processing to complete.

The above mechanism is also referred to as "trickle ICE".

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The Interactive Connectivity Establishment (ICE) protocol [RFC5245] describes mechanisms for gathering, candidates, prioritizing them, choosing default ones, exchanging them with the remote party, pairing them and ordering them into check lists. Once all of the above have been completed, and only then, the participating agents can begin a phase of connectivity checks and eventually select the pair of candidates that will be used in the following session.

While the above sequence has the advantage of being relatively straightforward to implement and debug once deployed, it may also prove to be rather lengthy. Gathering candidates or candidate harvesting would often involve things like querying STUN [RFC5389] servers, discovering UPnP devices, and allocating relayed candidates at TURN [RFC5766] servers. All of these can be delayed for a noticeable amount of time and while they can be run in parallel, they still need to respect the pacing requirements from [RFC5245], which is likely to delay them even further. Some or all of the above would also have to be completed by the remote agent. Both agents would next perform connectivity checks and only then would they be ready to begin streaming media.

All of the above could lead to relatively lengthy session establishment times and degraded user experience.

The purpose of this document is to define an alternative mode of operation for ICE implementations, also known as "trickle ICE", where candidates can be exchanged incrementally. This would allow ICE agents to exchange host candidates as soon as a session has been initiated. Connectivity checks for a media stream would also start as soon as the first candidates for that stream have become available.

Trickle ICE allows reducing session establishment times in cases where connectivity is confirmed for the first exchanged candidates (e.g. where the host candidates for one of the agents are directly reachable from the second agent). Even when this is not the case, running candidate harvesting for both agents and connectivity checks all in parallel allows to considerably reduce ICE processing times.

It is worth pointing out that before being introduced to the IETF, trickle ICE had already been included in specifications such as XMPP Jingle [XEP-0176] and it has been in use in various implementations and deployments.

In addition to the basics of trickle ICE, this document also describes how support for trickle ICE needs to be discovered, how

regular ICE processing needs to be modified when building and updating check lists, and how trickle ICE implementations should interoperate with agents that only implement [RFC5245] processing.

This specification does not define usage of trickle ICE with any specific signalling or media description protocol, contrary to [RFC5245] which defined a usage for ICE wht SIP and SDP. Such usages would have to be specified in separate documents.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This specification makes use of all terminology defined by the protocol for Interactive Connectivity Establishment in [RFC5245].

Vanilla ICE: The Interactive Connectivity Establishment protocol as defined in [RFC5245].

Candidate Harvester: A module used by an ICE agent to obtain local candidates. Candidate harvesters use different mechanisms for discovering local candidates. Some of them would typically make use of protocols such as STUN or TURN. Others may also employ techniques that are not referenced within [RFC5245]. UPnP based port allocation and XMPP Jingle Relay Nodes [XEP-0278] are among the possible examples.

3. Incompatibility with Standard ICE

The ICE protocol was designed to be fairly flexible so that it would work in and adapt to as many network environments as possible. It is hence important to point out at least some of the reasons why, despite its flexibility, the specification in [RFC5245] would not support trickle-ICE.

[RFC5245] describes the conditions required to update check lists and timer states while an ICE agent is in the Running state. These conditions are verified upon transaction completion and one of them stipulates that:

> If there is not a pair in the valid list for each component of the media stream, the state of the check list is set to Failed.

This could be a problem and cause ICE processing to fail prematurely in a number of scenarios. Consider the following case:

o Alice and Bob are both located in different networks with Network Address Translation (NAT). Alice and Bob themselves have different address but both networks use the same [RFC1918] block.
o Alice sends Bob the candidate 10.0.0.10 which also happens to correspond to an existing host on Bob's network.
o Bob creates a check list consisting solely of 10.0.0.10 and starts checks.
o These checks reach the host at 10.0.0.10 in Bob's network, which responds with an ICMP "port unreachable" error and per [RFC5245] Bob marks the transaction as Failed.

At this point the check list only contains Failed candidates and the valid list is empty. This causes the media stream and potentially all ICE processing to Fail.

A similar race condition would occur if the initial offer from Alice only contains candidates that can be determined as unreachable (per [I-D.keranen-mmusic-ice-address-selection]) from any of the candidates that Bob has gathered. This would be the case if Bob's candidates only contain IPv4 addresses and the first candidate that he receives from Alice is an IPv6 one.

Another potential problem could arise when a non-trickle ICE implementation sends an offer to a trickle one. Consider the following case:

o Alice's client has a non-trickle ICE implementation
o Bob's client has support for trickle ICE.
o Alice and Bob are behind NATs with address-dependent filtering [RFC4787].
o Bob has two STUN servers but one of them is currently unreachable

After Bob's agent receives Alice's offer it would immediately start connectivity checks. It would also start gathering candidates, which would take long because of the unreachable STUN server. By the time Bob's answer is ready and sent to Alice, Bob's connectivity checks may well have failed: until Alice gets Bob's answer, she won't be able to start connectivity checks and punch holes in her NAT. The NAT would hence be filtering Bob's checks as originating from an unknown endpoint.

4. Detecting Support for Trickle ICE

In order to avoid interoperability problems such as those described in Section 3, it is important that, before generating an offer and sending its first candidates an agent SHOULD first verify whether its correspondent also supports trickle ICE.

The exact mechanisms that would allow for such verifications are outside the scope of this document and should be handled by the

signalling protocol that is employing ICE.

Examples of how some signalling protocols already handle service and capabilities discovery include:

- o Service discovery [XEP-0030] and Entity capabilities [XEP-0115] for XMPP
- o Indicating User Agent Capabilities [RFC3840] for SIP

Usages of trickle ICE SHOULD make use of these mechanisms where they exist.

Also, in some cases it would be possible for an application to just "know" that support would be present. One example for this would be a WebRTC application that does not need to interoperate with applications from other web sites. Such applications can just enable trickle ICE without performing any additional checks.

In other cases yet, agents may choose to just send an offer that the remote party would reject as invalid unless it supports trickling. One such example would be an offer with no ICE candidates and an invalid default address (e.g. 0.0.0.0).

Usages of trickle ICE MUST define a way for offers or answers transporting the initial list of ICE candidates to indicate support for trickling. Note that an offer or an answer may indicate lack of support for trickle ICE even if other mechanisms have allowed to confirm that the remote agent does support it. In such cases agents should act as if trickle ICE is not supported for this particular session.

5. Sending the Initial Offer

An agent starts gathering candidates as soon as it has an indication that communication is imminent (e.g. a user interface cue or an explicit request to initiate a session). However, contrary to vanilla ICE, implementations of trickle ICE do not need to gather candidates in a blocking manner, strictly preceding the generation and transmission of their initial offer.

Trickle ICE agents MAY include any set of candidates in their initial offer. This includes the possibility of generating an offer with no candidates, or one that contains all the candidates that the agent is planning on using in the following session.

For optimal performance, it is RECOMMENDED that an initial offer contains host candidates only. This would allow both agents to start gathering server reflexive, relayed and other non-host candidates

simultaneously, and it would also enable them to begin connectivity checks.

If the privacy implications of revealing host addresses are a concern, agents MAY generate an initial offer that contains no candidates and then only trickle candidates that do not reveal host addresses (e.g. relayed candidates).

Prior to actually sending an offer, agents SHOULD verify if the remote party supports trickle ICE. If absence of such support is confirmed agents SHOULD fall back to using vanilla ICE or abandon the entire session.

All trickle ICE offers MUST indicate support of this specification. The exact means of providing this indication is left to the usages that define how signalling protocols employ trickle ICE.

Calculating priorities and foundations, as well as determining redundancy of candidates work the same way they do with vanilla ICE.

6. Receiving the Initial Offer

When an agent receives an initial ICE-enabled offer, it will check if the offerer supports trickle ICE as explained in Section 4. If this is not the case, the agent MUST process this offer according to the [RFC5245] procedures or standard [RFC3264] processing in case no ICE support is detected at all.

If, the offer does indicate support for trickle ICE, the agent will determine its role, start gathering and prioritizing candidates and, while doing so it will also send an answer, in order to start forming check lists and begin connectivity checks.

6.1. Sending an answer

The agent can create and send an answer at any point while gathering candidates. Just as with offers, answers can contain no or all candidates an agent is planning on using. Again, as with offers, it is RECOMMENDED that answers contain host candidates so that the remote party can also start forming checklists and performing connectivity checks.

The answer MUST indicate support for trickle ICE as described by usage specifications.

6.2. Forming check lists and beginning connectivity checks

After sending an answer, and as soon as they have gathered any candidates, agents will begin forming candidate pairs, computing their priorities and creating check lists according to the vanilla ICE procedures described in [RFC5245]. Obviously in order for candidate pairing to be possible, it would be necessary that both the offer and the ensuing answer contained candidates. If this was not the case agents will still create the check lists (so that their Active/Frozen state could be monitored and updated) but they will only populate them once they have learned any local and remote candidates.

Initially, all check lists will have their Active/Frozen state set to Frozen.

Trickle ICE agents will then also attempt to unfreeze the check list for the first media stream (i.e. the first media stream that was reported to the ICE implementation from the using application). If this checklist is still empty however, agents will continue examining media streams in the order they were reported and will unfreeze the first non-empty checklist.

Respecting the order in which lists have been reported to an ICE implementation, or in other words, the order in which streams had been described by the signalling protocol (e.g. SDP), is necessary so that checks for the same media stream would be performed simultaneously by both agents.

7. Receipt of the Initial Answer

When receiving an answer, agents will follow vanilla ICE procedures to determine their role and they would then form check lists and begin connectivity checks as described in Section 6.2.

8. Performing Connectivity Checks

For the most part, trickle ICE agents perform connectivity checks following vanilla ICE procedures. Of course, the asynchronous nature of candidate harvesting in trickle ICE would impose a number of changes:

8.1. Check List and Timer State Updates

The vanilla ICE specification requires that agents update check lists and timer states upon completing a connectivity check transaction.

During such an update vanilla ICE agents would set the state of a check list to Failed if the following two conditions are satisfied:

- all of the pairs in the check list are either in the Failed or Succeeded state;
- if at least one of the components of the media stream has no pairs in its valid list.

With trickle ICE, the above situation would often occur when candidate harvesting and trickling are still in progress and it is perfectly possible that future checks will succeed. For this reason trickle ICE agents add the following conditions to the above list:

- all candidate harvesters have completed and the agent is not expecting to learn any new candidates;
- the remote agent has sent an end-of-candidates message for that check list as described in Section 9.1.

Vanilla ICE requires that agents then update all other check lists, placing one pair in each of them into the Waiting state, effectively unfreezing the check list. Given that with trickle ICE, other check lists may still be empty at that point, a trickle ICE agent SHOULD also maintain an explicit Active/Frozen state for every check list, rather than deducing it from the state of the pairs it contains. This state should be set to Active when unfreezing the first pair in a list or when that couldn't happen because a list was empty.

9. Learning and Sending Additional Local Candidates

After an initial offer has been sent or received, agents will most likely continue discovering new local candidates as STUN, TURN and other non-host candidate harvesting mechanisms begin to yield results. Whenever such a new candidate is learned agents will compute its priority, type, foundation and component id according to normal vanilla ICE procedures.

The new candidate is then checked for redundancy against the existing list of local candidates. If its transport address and base match those of an existing candidate, it will be considered redundant and will be ignored. This would often happen for server reflexive candidates that match the host addresses they were obtained from (e.g. when the latter are public IPv4 addresses). Contrary to vanilla ICE, trickle ICE agents will consider the new candidate redundant regardless of its priority. [TODO: is this OK? if not we need to check if the existing candidate was already used in conn checks, cancel them, and then restart them with the new candidate ... and in this specific case there's probably no point to do that].

Then, if no remote candidates are currently known for this same stream, the new candidate will simply be added to the list of local candidates.

Otherwise, if the agent has already learned of one or more remote candidates for this stream and component, it will begin pairing the new local candidates with them and adding the pairs to the existing check lists according to their priority. Forming candidate pairs will work the way it is described by the vanilla ICE specification. Actually adding the new pair to a check list however, will happen according to the rules described below.

If the new pair's local candidate is server reflexive, the server reflexive candidate MUST be replaced by its base before adding the pair to the list. Once this is done, the agent examines the check list looking for another pair that would be redundant with the new one. If such a pair exists and its state is:

Succeeded: the newly formed pair is ignored.

Frozen or Waiting: the agent chooses the pair with the higher priority local candidate, places it in the state that the old pair was in (i.e. Frozen or Waiting) and removes the other one as redundant.

Failed: the agent chooses the pair with the higher priority local candidate, places it in the Waiting state and removes the other one as redundant.

In-Progress: The agent cancels the in-progress transaction (where cancellation happens as explained in Section 7.2.1.4 of [RFC5245]), then it chooses the pair with the higher priority local candidate, places it in the Waiting state and removes the other one as redundant.

For all other pairs, including those with a server reflexive local candidate that were not found to be redundant:

- if this check list is Frozen then the new pair will also be assigned a Frozen state.
- else if the check list is Active and it is either empty or contains only candidates in the Succeeded and Failed states, then the new pair's state is set to Waiting.
- else if the check list is non-empty and Active, then the new pair state will be set to

  Frozen: if there is at least one pair in the list whose foundation matches the one in the new pair and whose state is neither Succeeded nor Failed (eventually the new pair will get unfrozen after the the on-going check for the existing pair concludes);

Waiting: if the list contains no pairs with the same foundation as the new one, or, in case such pairs exist, they are all in either the Succeeded or Failed states.

9.1. Announcing End of Candidates

Once all candidate harvesters for a specific media stream complete, or expire, the agent MUST generate an "end-of-candidates" event for that stream and send it to the remote agent via the signalling channel. This would allow the remote agent to begin updating check list states and, in case valid pairs do not exist for every component in every media stream, determine that ICE processing has failed.

An agent MAY also choose to generate an "end-of-candidates" event before candidate harvesting has actually completed, if the agent determines that harvesting has continued for more than an acceptable period of time.

Once the agent sends the end-of-candidates event, it SHOULD update the state of the corresponding check list as explained in section Section 8.1

[TODO: should we also have an end-of-candidates for the entire harvesting process (as opposed to that of a single stream)]

10. Receiving Additional Remote Candidates

At any point of ICE processing, a trickle ICE agent may receive new candidates from the remote agent. When this happens and no local candidates are currently known for this same stream, the new remote candidates are simply added to the list of remote candidates.

Otherwise, the new candidates are used for forming candidate pairs with the pool of local candidates.

Once the remote agent has completed candidate harvesting, it will send an "end-of-candidates" event. Upon receiving such an event, the local agent MUST update check list states as per Section 8.1. This may lead to some check lists being marked as Failed.

11. Concluding ICE Processing with Trickle ICE

Trickle ICE processing SHOULD be concluded as explained in Section 8 of [RFC5245].

12. Interaction with non-Trickle ICE implementations

Trickle ICE implementations MUST behave as non-trickle and follow [RFC5245] unless they can confirm that the remote party supports this specification. [TODO: anything else?]

13. Security Considerations

[TODO]

14. Open Issues

At the time of writing of this document the authors have no clear view on how and if the following list of issues should be address here:

1. FILL IN

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

15.2. Informative References

[I-D.keranen-mmusic-ice-address-selection] Keranen, A. and J. Arkko, "Update on Candidate Address Selection for Interactive Connectivity Establishment (ICE)", draft-keranen-mmusic-ice-address-selection-01 (work in progress), July 2012.

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[XEP-0030] Hildebrand, J., Millard, P., Eatmon, R., and P. Saint-Andre, "XEP-0030: Service Discovery", XEP XEP-0030, June 2008.

[XEP-0115] Hildebrand, J., Saint-Andre, P., Troncon, R., and J. Konieczny, "XEP-0115: Entity Capabilities", XEP XEP-0115, February 2008.

[XEP-0176] Beda, J., Ludwig, S., Saint-Andre, P., Hildebrand, J., Egan, S., and R. McQueen, "XEP-0176: Jingle ICE-UDP Transport Method", XEP XEP-0176, June 2009.

[XEP-0278] Camargo, T., "XEP-0278: Jingle Relay Nodes", XEP XEP-0278, June 2011.

Authors' Addresses

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Phone: +1 650 678 2350
Email: ekr@rtfm.com

Justin Uberti
Google
747 6th St S
Kirkland, WA 98033
USA

Phone: +1 857 288 8888
Email: justin@uberti.name

Emil Ivov
Jitsi
Strasbourg 67000
France

Phone: +33 6 72 81 15 55
Email: emcho@jitsi.org